3.1 Use Cases

Subject/Topic/Focus:
- Introduction to Use Cases

Summary:
- User Goals and System Interactions
- Human and System Actors
- **Extends** Relationship between Use Cases
- **Uses** Relationship between Use Cases
- Usage of Use Cases in Objectory

Literature:
- Fowler
- Jacobson

---

Role of Use Case Diagrams in UML

![Diagram showing the role of use case diagrams in UML]

- **Requirements Documents**: actors and use cases
- **Use Case Diagrams**: workflows (interclass behavior), scenarios, object structures
- **Activity Diagrams**, **Interaction Diagrams**, **Class Diagrams**
Use Cases

Use cases represent **typical** sets of **scenarios** that help to **structure**, **relate** and **understand** the essential **requirements**.

- They come from traditional development methods and are adapted to OOAD.
- A use case is a related set of **typical interactions** between a user and a computer system.
  - “Make some text bold.”
  - “Create an index.”
  - “List all customers of a certain purchase item.”
- A use case captures some **action** on the application level.
- A use case achieves a discrete **goal** for the user.

To obtain use cases interview the users and ask them about the various things they want to do with the system.

Describe each use case in a **paragraph** and annotate it with documents, forms, ...

User Goals and System Interactions (1)

- The task of a use case can differ according to the point of view:
  - **Application domain level**: to reflect the **goal** the **user** wants to achieve.
  - **System design level**: to capture the required **system interactions**.
- Use cases may be small or large.

**Application domain level**
- Ensure consistent formatting for a document.
- Format two documents in the same style.

**System design level**
- Define a style, change a style and apply a style.
- Import a style from another, already existing document.
User Goals and System Interactions (2)

- This dichotomy between user goals and system interactions is present in most, but not in all situations, e.g.,
  - “Create an index.” is the same for a user goal and a system interaction.

- Importance of User Goals
  - for finding alternative ways to satisfy goals and
  - to capture all the requirements for user interfaces.

- Importance of System Interactions
  - for system structuring purposes and
  - for system functionality design.

Actors

- An actor is a role that a user or other system plays with respect to the system to be developed.

- Typical examples for actors:
  - Trading manager
  - Trader
  - Salesperson
  - Accounting System

- A single actor in a use case diagram can represent multiple users (or systems).

- A single user (or system) also may play multiple roles.

- Actors don't need to be human, e.g., an external system that needs some information from the current system is also an actor.
**Cases**

- The "case aspect" in a use case represents a high-level description of a desired action in which the actor is involved.
- It contains the information of the initial documents from which by gradual refinement further diagrams, for example, class diagrams, activity diagrams or interaction diagrams, are derived.

**Actors and Cases**

Introduced by [Jacobson 1994].

- **Trading Manager**
  - Set Limits
  - Analyze Risk <<uses>>
  - Price Deal <<uses>>
  - Limits Exceeded

- **Trader**
  - Capture Deal <<extends>>
  - Valuation

- **Sales Person**
  - Update Accounts

<<uses>>
<<extends>>
Interaction with External Systems

For the interaction with external systems there are four approaches:

1. Show all interactions with the remote system on the diagram.

2. Show external-interaction use cases only when it is the other system that initiates contact.

3. Show system actors only when they are the ones who need the use case.

4. Do not treat systems as actors at all but use the user requests instead.

Use external events to identify use cases which are not captured by actors. Think about all the possible events from the outside world to which you want to react.

Working with Use Cases

- Use cases are all about externally-required functionality.
  - If the Accounting System needs a file from the system under development, this is a requirement that needs to be satisfied.
- Discuss use cases with system users.
  - What are the real user goals? Consider alternative ways to meet those user goals.
- Adjust the granularity of your use cases according to the complexity of the individual problem you are working on, bearing in mind that
  - too many use cases can be overwhelming while
  - too few use cases may hide important details.
- There are several possibilities to carry out a use case, therefore use cases can have many realizations (manual, semi-computerized, fully automatic; different designs, ...). This is a design issue!
**Extends (1)**

Use the `extends` relationship when you have a use case that is similar to another use case but does a bit more.

- In this example the basic use case is `Capture deal`.
- In case certain limits for a deal are exceeded the additional use case `Limits exceeded` is performed.

**Extends (2)**

- Extensions are used instead of modeling every extra case by a single use case.
- Extensions are used instead of creating a complex use case that covers all variations.
- Extensions mean “this use case is similar to that use case with the exception of ...”.
- How do you address case variation?
  - Capture the simple, normal use case first.
  - For every step in that use case ask: „What could go wrong here?“
  - Plot all variations as extensions of the given use case.
- Don’t be surprised if there is a fairly high number of extensions.
- Listing the extensions separately makes things easier to understand.
- Splitting of use cases into extensions can be done during elaboration or construction.
Uses

- The **uses** relationship occurs when you have a chunk of behavior that is similar across several use cases.
- Use **uses** when you are repeating yourself in two or more separate use cases.
- Copying the description of that behavior introduces redundancy and may lead to inconsistencies when the behavior is changed.

![Diagram](Diagram1)

Uses and Extends

- The similarities between **extends** and **uses** are that they both imply factoring out common behavior from several use cases to a single use case that is
  - used by several other use cases or
  - extended by other use cases.
- From the **actor’s viewpoint**
  - **extends** means, both the normal use case and the extension are performed by the actor.
  - **uses** means, there is often no actor associated with the common use case.
- Apply the following rules:
  - Use **extends**, when you are describing a variation on normal behavior.
  - Use **uses** when you want to split off repeating details in a use case.
Scenarios, Workflow, Object Structures

- A certain path through a use case is called a scenario. A scenario shows a particular set and combination of conditions within that use case.
  
  E.g., ordering some goods:
  
  - Scenario 1: All goes well.
  - Scenario 2: There are not enough goods.
  - Scenario 3: Our credits are insufficient.

- Processes involving different use cases are shown in workflows, e.g., from ordering to delivery and payment.

- The structure of the objects (and actors) that occur in use cases are described by classes, e.g., customers, goods, invoices.

Use Cases in Objectory

- Use cases are an essential tool in requirements capturing and in planning and controlling an iterative project.
- Capturing use cases is one of the primary tasks during the elaboration phase.
- UML delivers the use cases to analyze the requirements of a system.
  
  - Use cases are typical interactions a user has with the system.
  - A use case indicates a function that the user can understand and that has value for the user.
  - Use cases can vary considerably in size.

One of the most important things during elaboration is to discover all the potential use cases.
Use Cases: Examples

- For a person using a **database** a typical use case would be:
  - “list all customers who have ordered a certain product”
  - “create a list with my top 10 customers”
  - “I want fax-letters to be sent automatically”
- A developer responds with specific **cost estimates**:
  - “The top 10 customer list can be developed in a week.”
  - “Creating the auto-fax function will take two months.”
- User and developer negotiate about the **priorities**:
  - Developer: “I could start with the sold - products list.”
  - User: “I definitely need the top 10 customers list first.”

Planning: Use Cases & Iterations

- The essence of a plan is to set up a series of **iterations** for **construction** and to assign **use cases** to iterations.
- The plan is finished, when each use case is put into an iteration and when for each iteration **start/delivery dates** are scheduled.
Planning: Categorize Use Cases

- The users should indicate the level of priority for each use case.
  - “I absolutely must have this function for any real system.”
  - “I can live without this function for a short period.”
  - “It is an important function, but I can survive without it for a while.”

- The developers should consider the architectural risk.
  - Do not omit use cases which later cause you to do a lot of rework to fit them in.
  - Concentrate to the use cases which are technologically most challenging.

- The developers should be aware of the schedule risks.
  - “I’m pretty sure I know how long it will take.”
  - “I can estimate the time only to the nearest man-month.”
  - “I have no idea.”